INFO-F-404 : Real-Time Operating Systems
Project 1 : Audsley

Rémy Detobel, Stanislas Gueniffey and Denis Hoornaert

December 21, 2017

1 Implementation choices

The aim of the present section is to give to the reader a quick overview of the utility of each of the implemented classes.

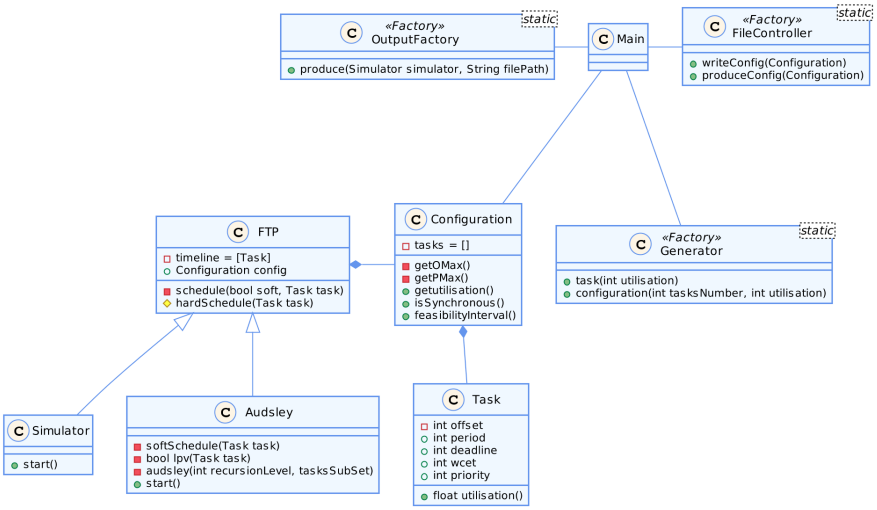

Figure 1: Pseudo-uml of the project

The project is composed of various classes. Among these classes, some are elementary as they are used in every aspect of the project (e.g. `Configuration`) whereas the others are specific to a feature (e.g. `Generator`).

The "elementary" classes :

- `Configuration` : This object is produce by the `FileController` class and, similarly to the theory, is mainly a set/collection of tasks. The class also provides useful methods used throughout the project by other classes (e.g. `isSynchronous()`).
- `Tasks` : This class can simply be seen as a bunch of useful data (close from the theory : nothing less-nothing more).

The "scheduling dedicated" classes :

- `FTP` : Abbreviation of Fixed Task Priority, this class just regroup common features of its two children (`Simulator` and `Audsley`). Typically, these features are the time line structure, the hard scheduling and so on.
- `Simulator` : The class has for only objective to hard schedule every tasks of a given `Collection` object and to print the "events" of the simulation as stipulated in the assignment statement.

- `Audsley` : Even though this class inherits from `FTP` like `Simulator`, its objective is different as it does not aim to simulate anything. In fact, this class is focused on computing the *least priority viable task* of a tasks collection and deducing a adequate priority assignment using the Audsley algorithm.

There are three Factory classes :

- `FileController` : This class is dedicated to the reading and the writing of tasks collection from and to a file.
- `OutputFactory` : The aim of this class is to encapsulate the mechanism used to generate a visual representation of a scheduling (For further information see 3).
- `Generator` : As ask in the assignment statement, the project should be able to generate a task collection given two parameters. IN our project, the class `Generator` is the class that achieves this feature.

2 Difficulties encountered

One of the difficulties encountered during the realisation of the project is the generation of random systems. More precisely : how to ensure that the utilisation of the generated system is as close as possible to the utilisation specified by the user ?
The issue has been addressed as follow :
By definition, the total utilisation of the system is given by :

$$U(\tau) = \sum_{i}^{n} U(\tau_i) \qquad\qquad U(\tau_i) = \frac{C_i}{T_i}$$

Where :

- n is the number of tasks specified by the user
- $U(\tau)$ is the total utilisation of the system specified by the user

We generate a vector of size n for which each cell contains a random integer $i \in [10, 100]$. This vector is supposed to be the vector of tasks utilisation. Thus, we compute the factor by which each cell has to be divided so that the sum is equal to the desired utilisation.

$$U(\tau) = \frac{\sum_i^n rand(i)}{factor} \qquad \Leftrightarrow \qquad factor = \frac{\sum_i^n rand(i)}{U(\tau)}$$

For example :

$$rand = [30, 40, 50, 60],\ |rand| = 4,\ U(\tau) = 70$$
$$factor = \frac{30 + 40 + 50 + 60}{70} \approx 2.57$$
$$utilisations = [11.67, 15.56, 19.46, 23.35]$$

Once done, we can create a task based on the given utilisation (one of the vector) as follow :

1. we generate a random offset
2. we generate a random WCET with the condition that it must be greater than 0 because otherwise the task is useless and because a negative wcet makes no sense
3. for the period, we cannot generate a fully random value as the utilisation must be respected. Hence, we determine its value using the following formula :

$$U(\tau_i) = \frac{C_i}{T_i} \qquad \Leftrightarrow \qquad T_i = \frac{C_i}{U(\tau_i)}$$

4. we generate a random deadline with the condition that it must at least be equal to the wcet and at most equal to the period

Applying this process, implies the use of a lot of float numbers and the approximations that comes with. However, we still manage to have a nice total utilisation approximation.

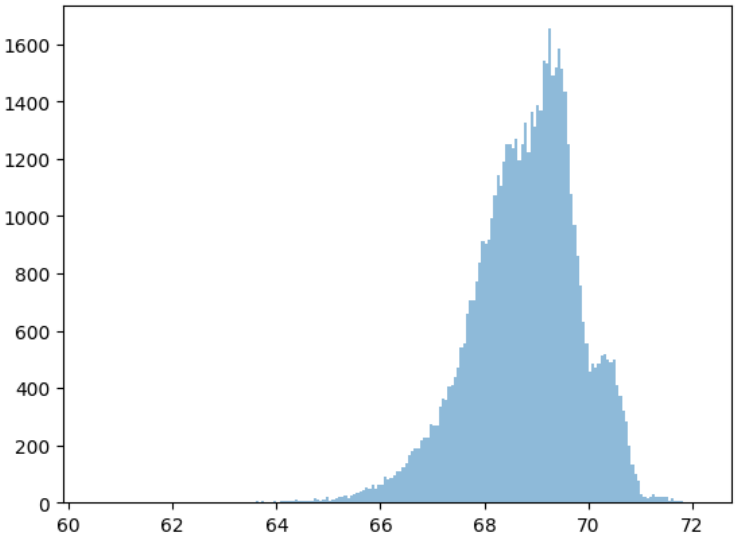

Figure 2: Distribution of utilisation of generated systems (6 tasks with a targeted 70% of utilisation)

3 Project output dependencies

The python modules used to generate the image of the scheduling are called `numpy` and `matplotlib`. Even though they are quite famous amongst python programmers, they are not part of the standard library. Hence, they need to be manually installed sometimes. This can be easily be achieved by using `pip3` and by typing the following commands :

```
sudo pip3 install numpy
sudo pip3 install matplotlib
```

The commands above have been tested on Ubuntu 16.04 LTS but might differ depending on the distribution and the version.

Important : The code will only produce an image if the user append the name of the desired image file to the command used for starting a simulation. If no fourth argument is provided then a simple simulation will be executed. Example :

```
python3 project.py sim <start> <stop> <taskFile> <imageFile>
```

Notice that if you run the project on the NO.4, trying to generate an image will result in a project crash as the library used is not installed on these computers.

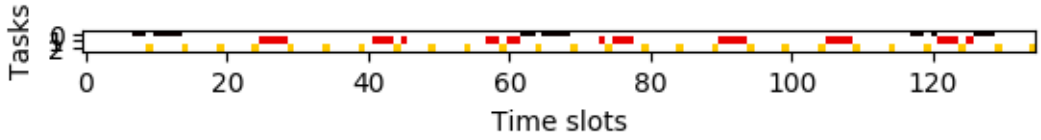

Figure 3: An example of an image generated by the project